# Agile Software Design

- Learn many of the common design patterns applied to algorithms and overall systems
- Discuss applying proper leadership practices to an agile team
- Learn how to avoid and pay down technical debt

This course serves to continue the exploration of key aspects of agile software programming practices begun with our agile programming course [1]. Attendees will have a chance to practice and dig deeper in relevant topics such as leadership styles, recurring development patterns, and methods for handling technical debt. A deeper dive on test automation, continuous delivery, and empirically proven approaches to the design process will build upon the skills of those practitioners eager to take the next step along their agile journey.

**Who Should Attend**
Developers, software developers in test, architects, and technical leads who have a basic understanding of good programming practices and want to take their analysis, design and programming skills to a new level.

**Laptop Required**
This class involves hands-on activities using sample software to better facilitate learning. Each student should bring a laptop with a remote desktop protocol (RDP) client preinstalled. Connection specifics and credentials will be supplied during class. Please verify permissions with your IT Admin before class. If you or your Admin have questions about the specific applications involved, contact our Client Support team.

**ICAgile Certification**
Successful attendees are awarded the ICAgile Certified Professional: Agile Software Design (ICP-ASD). Additionally, the certified professionals will be listed on the ICAgile website, indicating their designation. Coveros recommends From Fragile to Agile: Practical Approaches to Adopting Agile[2] for those seeking ICAgile certifications.The ICAgile certification fee is included with your registration for your convenience.

## Course Outline

**Architecture**
Conway's law
Scaled agile
Staffing projects
Choosing languages
Choosing frameworks
Your approach to dependencies

**Stakeholders**
Team members
Getting the best requirements

**Design**
Small design
Large design
Abstraction

**Automation**
Skipping the UI
UI testing
Non-functional testing

**Technical debt**
Costs of tech debt

**Technical leadership**
Entrepreneurship
Pairing or mobbing

**Continuous delivery**
The highest priority
Simple approach
Common approach

Bug backlogs

**Simplicity**